

## .NET Performance

**Duration: 4 days**

**Course Description:** This four-day instructor-led course provides students with the knowledge and skills to develop high-performance applications with the .NET Framework. Building high-performance applications with the .NET Framework requires deep understanding of .NET memory management (GC), type internals, collection implementation, and most important – tools for measuring application performance. The course features numerous performance measurement scenarios, optimization tricks, deep focus on .NET internals, and high-performance development guidelines.

**Audience:** This course is intended for C# developers with practical experience of at least a year with the .NET framework.

**Prerequisites:** Before attending this course, students must have: Working knowledge of C# 2.0 ,Working knowledge of the .NET Framework 2.0 - or - Completed course 2349 or have equivalent knowledge in the above topics. cursory familiarity with operating system topics: Threads, paging, FS cache. cursory familiarity with computer organization topics: CPU, cache, memory.

### Topics:

#### MODULE 1: PERFORMANCE MEASUREMENT

This module explains how to measure the performance of .NET applications on the Windows platform using a variety of automatic and manual tools.Lessons

- Managed Code Cost Model
- Profilers – CPU profilers, memory profilers, custom profiler API
- Manual Measurement – Performance Harness, performance counters, Windows Management Instrumentation (WMI)
- Static Code Analysis – Performance-related FxCop rules

#### Lab: Measuring Performance

- Measuring string concatenation with string and StringBuilder
- Measuring enumeration and summing of integers in a list compared to an array

After completing this module, students will be able to:

- Measure the performance of short code sections.
- Measure the performance of real applications using performance counters and other diagnostic tools.
- Pinpoint performance issues with memory management or CPU utilization using various profilers.
- Prevent performance problems before code execution by using static code analysis rules.

#### MODULE 2: SYSTEM DIAGNOSTICS

This module explains how to provide better debugging capabilities for user and framework code, how to obtain process and thread information including stack traces, and how to manually measure application performance.

Lessons

- Debugger Attributes – Controlling type debugger display, debugger type proxy, and debugger visualizers
- Viewing Processes and Threads

## .NET Performance

- Obtaining Stack Traces
- Measuring Time with a Stopwatch

After completing this module, students will be able to:

- Control debugger display of user and framework types to obtain an easier debugging experience.
- Obtain stack traces of threads and examine threads and processes.
- Measure application performance using the built-in .NET stopwatch.

### MODULE 3: TYPE INTERNALS – REFERENCE TYPES AND VALUE TYPES

This module explains how to use reference types and value types properly to improve application performance, how to decide whether virtual functions or interfaces are appropriate performance-wise, and how to least affect the CLR's internal handling of types from a performance perspective

- Review – Reference types and value types differences
- Type Implementation – Type Object Pointer, Sync Block Index
- Virtual Methods – Invoking virtual vs. non-virtual methods
- Value Type Methods – Equals and GetHashCode inherited from Object and ValueType

#### Lab: Implementing Value Types

- Implementing and measuring GetHashCode for value types

After completing this module, students will be able to:

- Properly choose between reference types and value types and appreciate

the performance characteristics of virtual vs. non-virtual method dispatch.

- Implement value types correctly.
- Implement object synchronization correctly by utilizing sync blocks.

### MODULE 4: GARBAGE COLLECTION

This module explains how to properly interact with the .NET garbage collector (a service responsible for automatically reclaiming unused memory) to maximize application performance.

- Comparing Garbage Collection Techniques – Reference counting, copying, tracing garbage collection
- Garbage Collection Requirements
- First Model: Managed Heap – Next object pointer, GC flavors, thread suspension
- Second Model: Generations
- Third Model: GC Segments
- Interacting with the GC – Managed and native code
- Finalization – Finalization, resurrection, Dispose pattern
- Weak References
- GC Best Practices

#### Lab: Diagnosing a Memory Leak

- Diagnosing a finalization-related memory leak

After completing this module, students will be able to:

- Properly interact with the garbage collector to improve application performance.
- Diagnose memory leaks and plumb them.
- Be better citizens of the operating system with regard to managed application memory requirements.

## .NET Performance

### MODULE 5: GENERICS

This module explains how to implement generic code and use generic collection classes to maximize application performance, specifically when value types are concerned.

- Generic Classes, Interfaces and Methods
- Generic Collections
- Implementation of Generics in Runtime – Reflection and generics, Java generics implementation, C++ templates implementation

#### Lab: Implementing a Generic Class

- Implementing a generic stack and measuring its performance

After completing this module, students will be able to:

- Implement generic classes, interfaces, and methods.
- Use generic collections to maximize application performance.

### MODULE 6: UNSAFE CODE AND C++/CLI

This module explains how to use unsafe code (pointers) in C# to maximize performance of low-level scenarios and how to use C++/CLI as a managed language offering performance benefits. Lessons

- Unsafe Code – Pointers, compilation with /unsafe, the fixed statement
- C++/CLI – Syntax basics, classes, delegates, generics

#### Lab: Implementing Memory Copy

- Implementing unsafe memory copying facilities and comparing performance to built-in approaches

After completing this module, students will be able to:

- Implement unsafe algorithms to maximize application performance in low-level scenarios.
- Use C++/CLI with interoperability and performance in mind.

### MODULE 7: COLLECTIONS

This module explains how to choose the right collection implementation with cache and page fault considerations in mind and how to implement collections properly.

- Collection Considerations – Choosing the right collection, CPU cache, paging
- Implementing Collections – Implementing interfaces properly, working around nlining limitations, and using automatic iterators with caution

After completing this module, students will be able to:

- Choose the proper collection for the task based on performance considerations.
- Implement collections for the maximum performance possible.